GENERALIZED APPROACH FOR EVALUATING DATA BASE ORGANIZATION AND INDEXING METHODS

Rodolfo Mendiola Clautero

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

GENERALIZED APPROACH FOR EVALUATING
DATA BASE ORGANIZATION AND INDEXING METHODS

by

Rodolfo Mendiola Clautero

June 1975

Thesis Advisor: N. F. Schneidewind

Approved for public release; distribution unlimited.

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Generalized Approach for Evaluating Data Base Organization and Indexing Methods		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; June 1975 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Rodolfo Mendiola Clautero		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE June 1975 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Data base organization

Data structures

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper is a study of data base organization and indexing methods with emphasis on the evaluation process. The approach of the study is focused on the data structure, a major characteristic of data base. Other aspects of the subject, indexing, for example, were discussed in relation to data structure. It was found that completeness and lucidity of knowledge of data base organization and indexing methods is

necessary if one is to do a good job of evaluating a data base system.

Generalized Approach for Evaluating Data Base Organization and Indexing Methods

by

Rodolfo Mendiola Clautero
Lieutenant Commander, Philippine Navy
B.S., Philippine Military Academy, 1966

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1975

ABSTRACT

This paper is a study of data base organization and indexing methods with emphasis on the evaluation process. The approach of the study is focused on the data structure, a major characteristic of data base. Other aspects of the subject, indexing, for example, were discussed in relation to data structure. It was found that completeness and lucidity of knowledge of data base organization and indexing methods is necessary if one is to do a good job of evaluating a data base system.

TABLE OF CONTENTS

I. INTRODUCTION

The subject of data base organization is receiving increased attention in the computer community. This is the case because data bases are used in almost all types of computer applications, e.g., business and scientific applications. The computer community's involvement in data base studies has been limited to specific applications. Recently, it was realized that application independent data base systems are important. Generalized data base system is the name used for this type of system.

Various groups are making efforts to organize the study of data base system. Foremost in the field is the CODASYL[1] Systems Committee [3, 4]. This committee has as its goal the development of specifications for a common language and functions. Concurrent with the initial publication of the CODASYL Systems Committee's report, arguments were raised as to the advantages and disadvantages of generalized data base systems as compared with the traditional approach of application dependent data base systems. To fully understand the different arguments in favor of or against an approach, one needs to delve deeper into the subject of data systems. This paper is an attempt to put into perspective the multiple and diversified data base system developments.

[1] CODASYL is the abbreviation of Conference on Data Systems Languages.

II. CLASSIFICATIONS OF DATA BASE MANAGEMENT SYSTEMS

Data base management systems have various characteristics. The CODASYL Systems Committee in its technical report dated May 1969 gave the following major data base characteristics: data structure class, storage structure class, generalized processes provided, language type, language form, modes of use, file media, hardware environment and operating systems environment. Almost all of these major characteristics have subclassifications, e.g., under generalized processes provided there are file definition, file creation, file updating and interrogation.

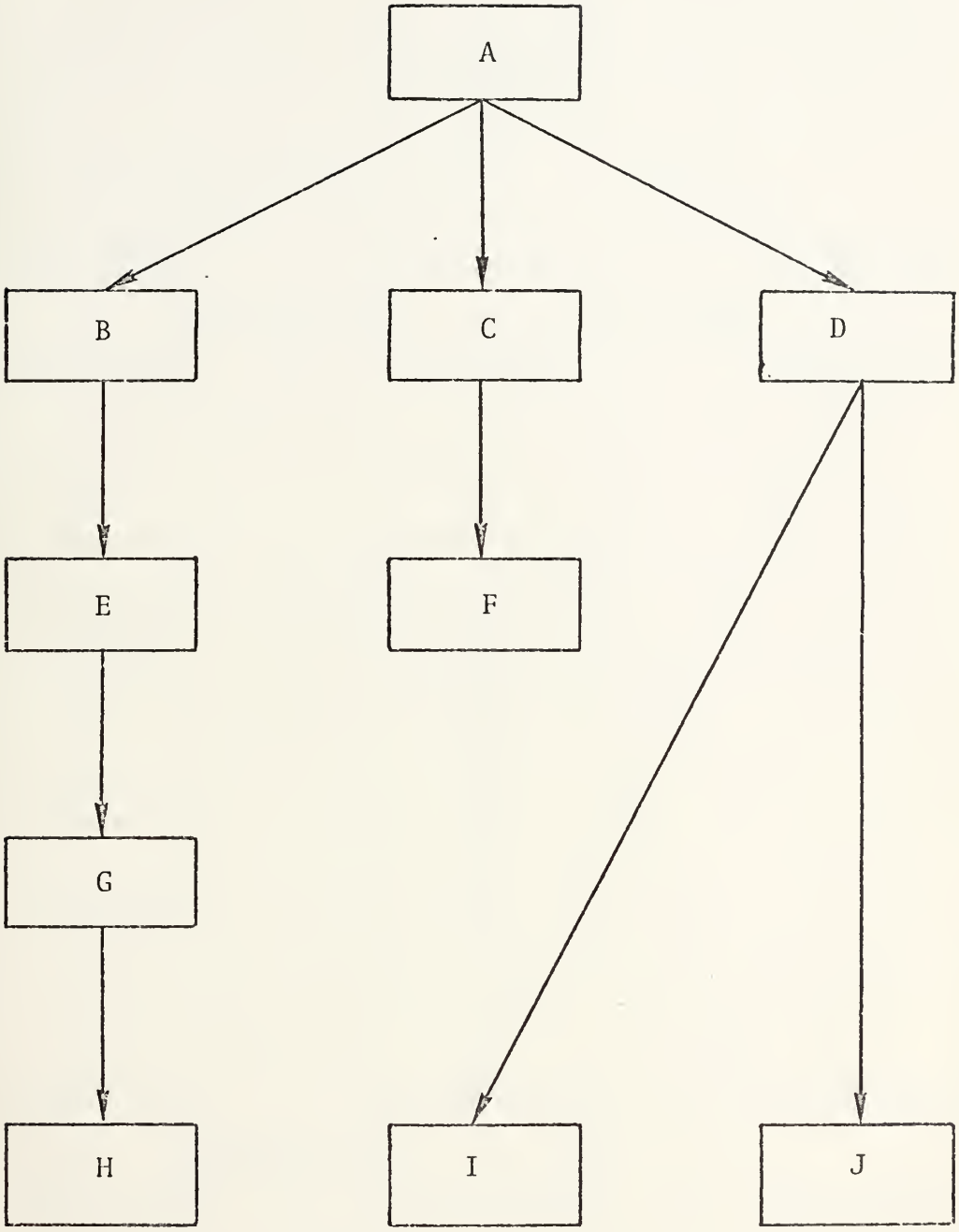
The ensuing discussion will deal with the more important areas of data organization characteristics. The system evaluation process will also be discussed as much as possible. A number of these characteristics are dependent. An example is the file media option. One would keep in mind physical storage characteristics when selecting a data organization. Also, the hardware and operating systems environment must be considered. Sometimes these facilities are fixed beforehand. In other cases, the computer system is selected in addition to the data base system.

A. DATA STRUCTURES

Data structure is the logical or hierarchical relationships among elements in the data base. In contrast, storage structure is the actual organization of data elements on the

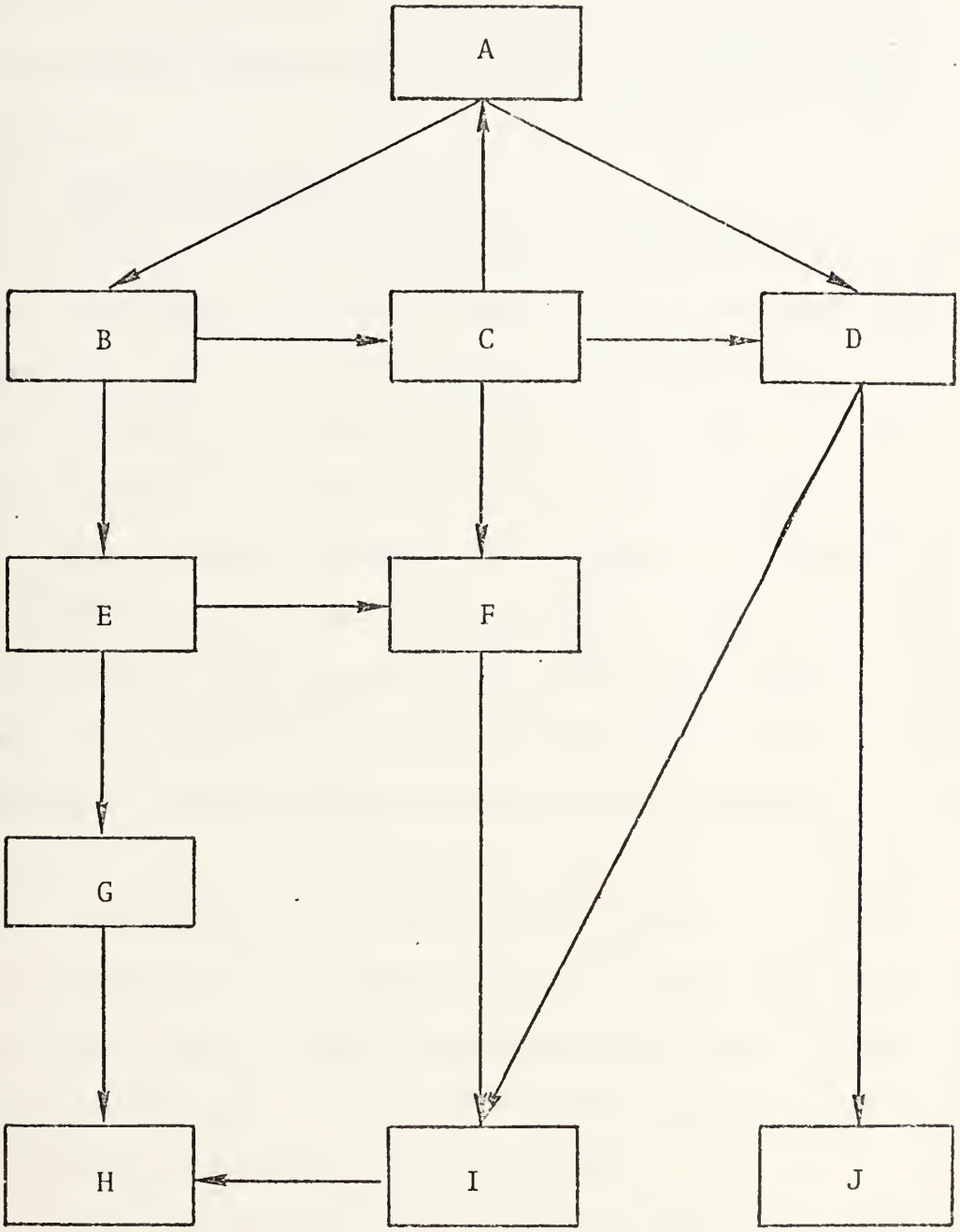
physical storage system. And since data relationships usually exist among records, data structure can further be viewed as the relationships among records. Based on this viewpoint, data structures are classified according to how the records are cross-referenced. There are three general classifications of data structures: (1) Multiple record file: In this classification no record "owns" any record. Records are related to each other only in their physical positions; hence, their relationships can be defined only by logical keys. For example, for records stored sequentially, a record's key may be smaller or greater than another record's key, depending on its position relative to the other record. (2) Hierarchical file: This type of data structure is sometimes called a tree structure because of the schematic representation of its data relationships. In this classification a record can "own" any number of records but it can be "owned" only by one record. For an example, in Figure 1, record D "owns" records I and J and is "owned" by record A only. (3) Network structure: In this classification a record can "own" or be "owned" by any number of records. For an example, in Figure 2, record D "owns" records I and J and is "owned" by both records A and C. [London, 1973].

Due to the advent of direct access storage technology, multiple record files are not as numerous as they once were. Thus, to a great extent, tree and network data structures are used in relating data within a file or from one file to another. Further, these data structures are found to be more



HIERARCHICAL OR THREE DATA STRUCTURE

Figure 1



NETWORK DATA STRUCTURE

Figure 2

efficient for many data base applications. Chains and chain-modifier ring structure and list structure are two main methods used in cross-referencing data in tree and network data structure.

1. Chain Data Structures

Chaining is a technique of linking logically-related records by means of pointers; pointers are special fields incorporated into each record structure which contain reference(s) to the next logically-related record(s). A group of records so structured and linked is called a chain.

Chaining has a number of advantages [London, 1973]. Data in chains can be accessed by several keys; the number of keys depends on the number of independent chains a record belongs to. The use of multiple record keys allows accesses to records. It also allows records to be related in a number of ways. Further, it allows data retrieval in a logical sequence regardless of the physical sequence of records. Another advantage of chaining comes to light when additional reports are required after the system had been installed. Chaining allows users to produce newly required reports without necessarily restructuring the file.

As to the disadvantages [London, 1973], the most critical is the cumbersome updating procedure which requires significant processing time. This disadvantage, however, could be decreased to a reasonable level by planning the system updating. Two main factors have to be watched in regard to system updating: the frequency of additions and

deletions of records in the chains and the frequency of data changes within the established chains. (The former is usually referred to as the volatility of data.) Other disadvantages of chaining are record growth and the length of chains. Search time to retrieve items from the chains is directly proportional to the size of the records and the length of the chains. Thus, although this disadvantage is less critical than the former, it is by no means unimportant.

2. List Data Structures

A list data structure is comparable to chaining and is conceptually an extension of chaining. However, instead of using pointers to link records, the list data structure technique uses list, such that each list of addresses (or records) contains logically related items only. Lists are used for the primary purpose of reducing the length of search time (the second disadvantage of chaining). A type of list is the simple list.

A simple list approach is used in conjunction with chains of records. If chain length is to be restricted or if the search time is to be reduced, the chains are divided into a number of segments, each of which owns an entry point. The table of these entry points is the list. The entry points are used as the primary keys in searching the file. Actual addresses can be used as entry points. Variations are possible in building lists. The primary keys may be made to hold more than an entry point to a chain if the chain is long.

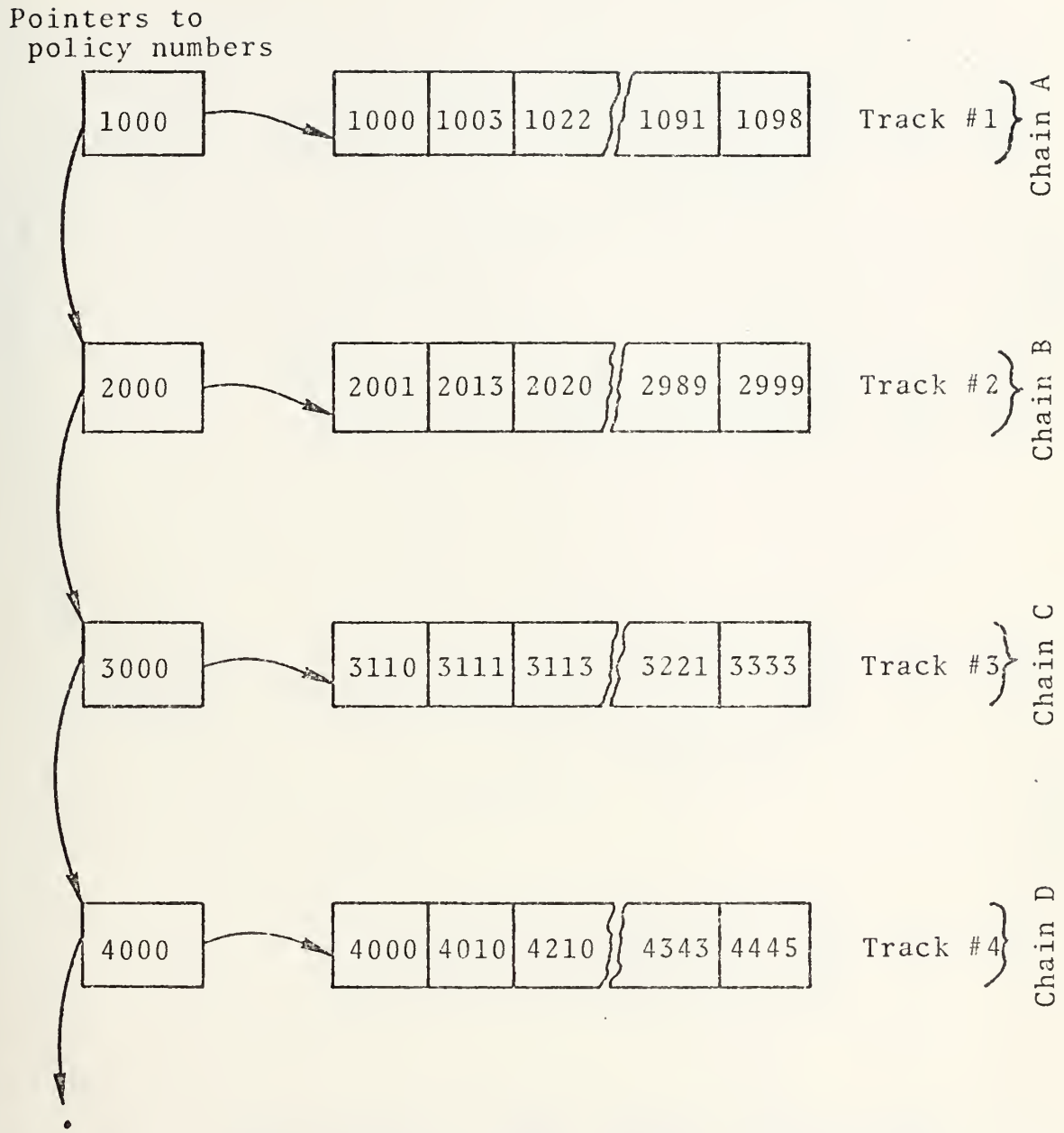
On the other hand, the number of primary keys can be reduced if the chains are relatively short.

Figures 3, 4, 5 and 6 are diagrams of a simple list and its variations. It is assumed that a storage drum would be used. The categorization of records per specific characteristic is termed an attribute. Further, in these figures it is assumed that the primary index would be composed of insurance policy numbers of car owners. Chains A, B, C, D, etc. are used to designate each thousand series of policy numbers.

In Figure 3, the number of records is on the average just enough to occupy one track of the storage drum. This facilitates the storage scheme of chains A, B, C, D, etc. Each chain could be stored on one track each.

If the average number of records related to each attribute is large, it would be necessary to store the records on more than one track. In this situation each primary index entry can be made to point to more than one track. This is what is done in Figure 4. The index entry of 1000 is used to point to track #1 and track #2. Also, the primary index entry of 2000 is used to point to track #3 and track #4. Note that pointers are needed from the last entries of track #1 and track #3 to the first entries of track #2 and track #4, respectively.

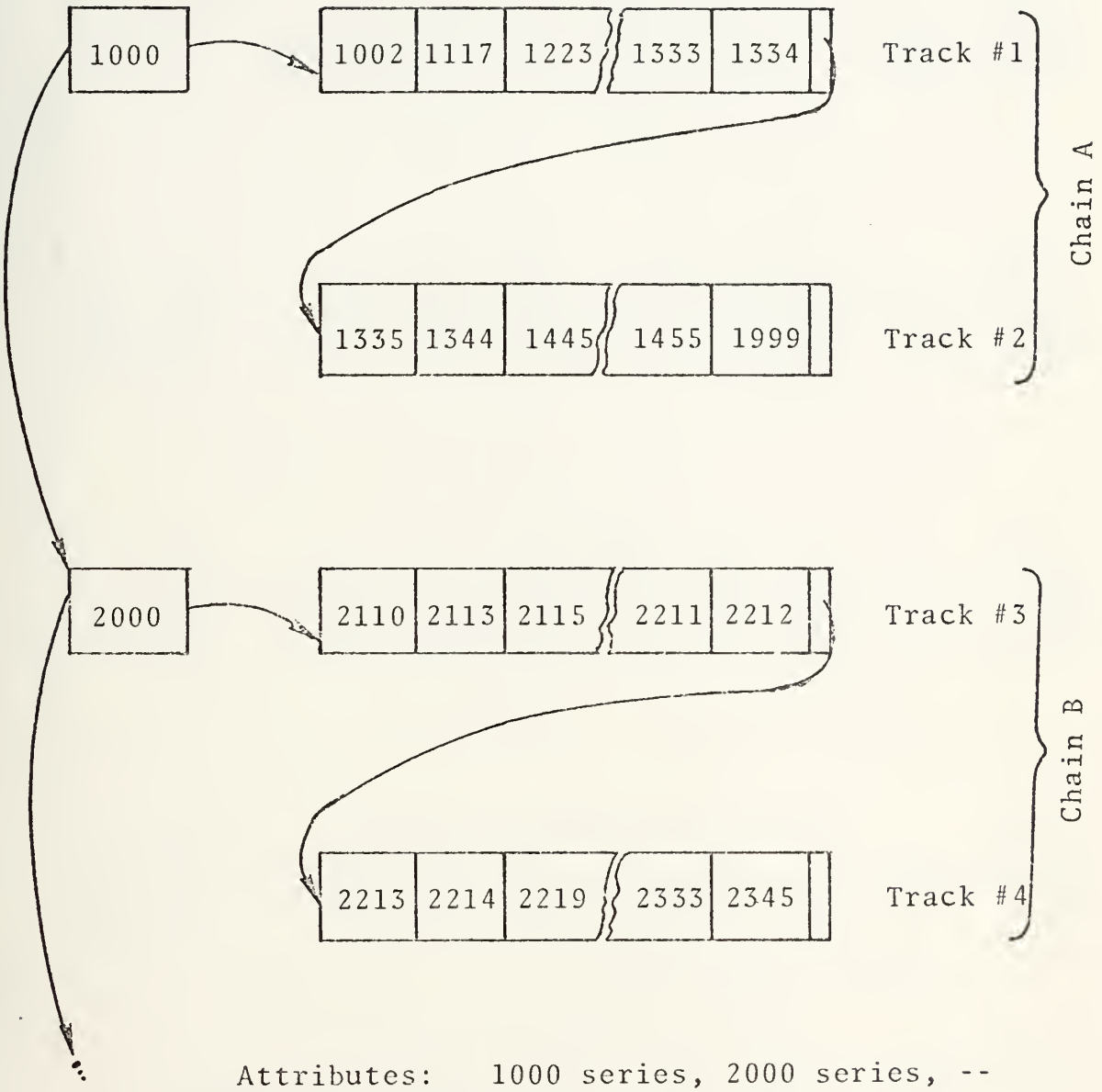
In Figure 5 is another list variation. The number of records per attribute is small and can be contained on some part of each track. Records related to other attribute(s)



Attributes: 1000 series, 2000 series, 3000 series, --
Primary keys: 1000, 2000, 3000, 4000, --
Entry points: 1000, 2001, 3110, 4000, --

SIMPLE LIST DATA STRUCTURE

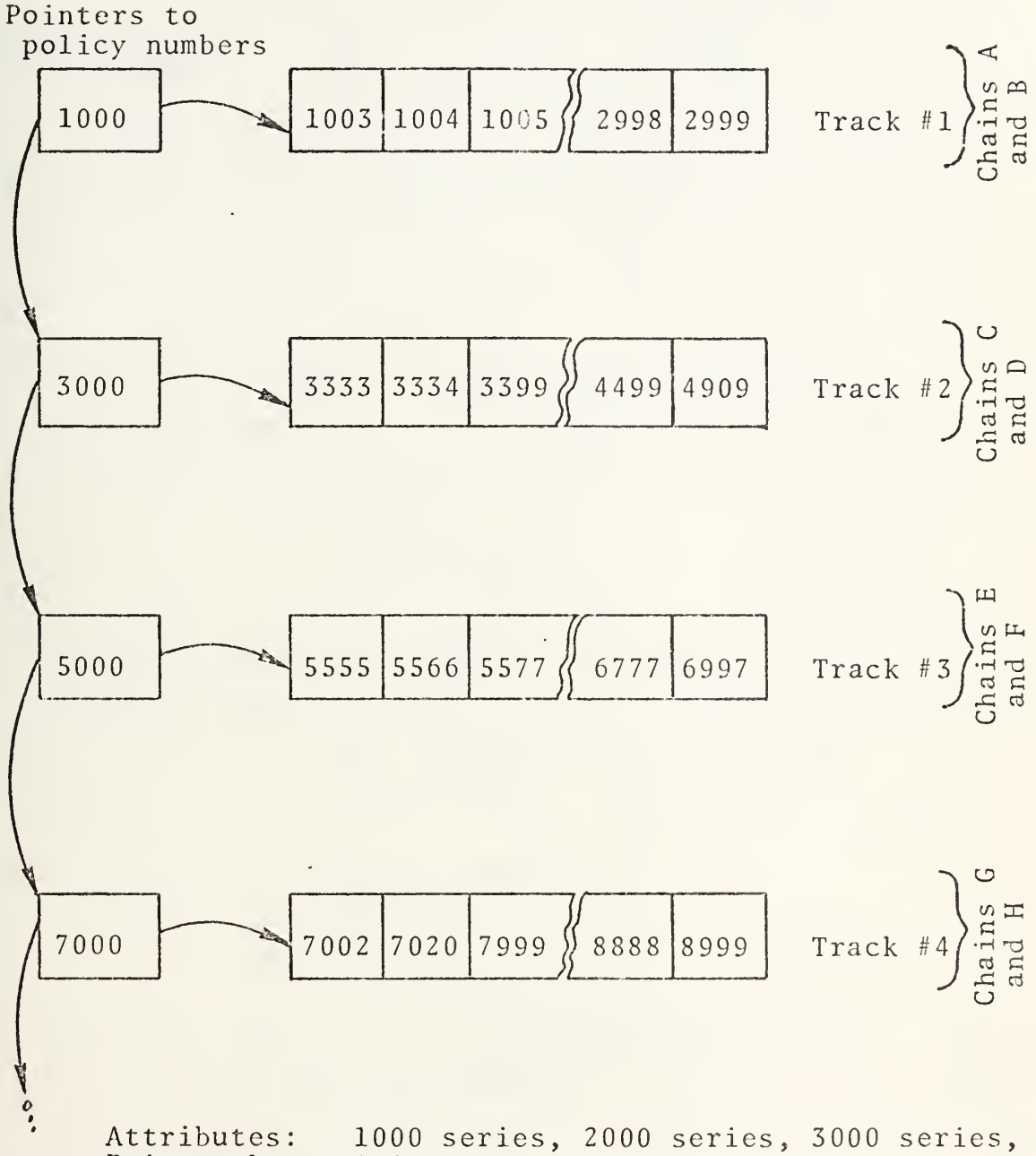
Figure 3



Attributes: 1000 series, 2000 series, --
Primary keys: 1000, 2000, --
Entry points: 1002, 2110, --

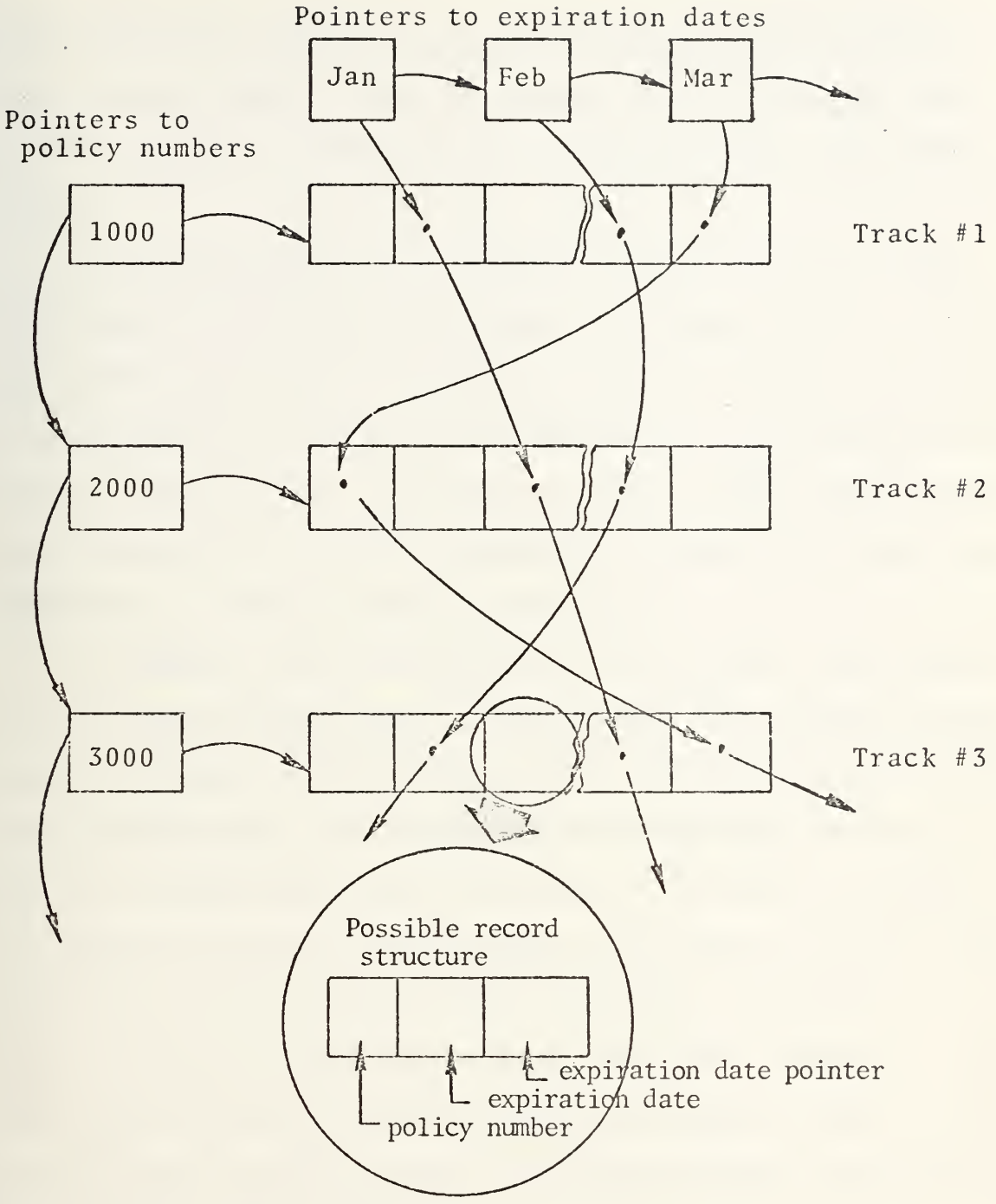
SIMPLE LIST DATA STRUCTURE VARIATION
(Primary keys hold more than one entry point)

Figure 4



SIMPLE LIST DATA STRUCTURE VARIATION

Figure 5



Primary index (main attribute) : 1000, 2000, 3000, --
Secondary index (second attribute): January, February, March, --

LIST DATA STRUCTURE WITH A SECOND DEGREE OF INVERSION

Figure 6

could still be stored in the unused space of the tracks to fully utilize them. Hence in Figure 5, the pointers could be made to point to every other attribute (1000, 3000, 5000, 7000).

Assume that we want another list (secondary index) that would be composed of insurance expiration dates. Since the records are made contiguous on the basis of policy numbers, we need pointers to follow the sequence of insurance expiration dates. This is shown in Figure 6. Since there would be two lists, the primary and the secondary indices, the scheme in Figure 6 is a second degree inversion.

Simple lists are extensions of chaining, hence they have the same disadvantages, except that the extended search time in chains is eliminated. The search capability of lists is of considerable use. Much of the searchings is done in the list, thus unnecessary retrieval of records is avoided. The degree of search effectiveness of a list is directly proportional to the degree of file inversion. Inversion of files or the inverted list will be discussed later. However, the simple list requires more storage space than chaining. Depending upon the application, the reduced search time may well be worth the additional space.

The extreme of list data structuring is the inverted list. This approach requires one entry for each type of attribute in the file; hence, the lists would hold the references between records. Conceptually, the inverted list is a series of lists of pointers to data records. The lists can be held

at the beginning of the file or at another location. The inverted list's record insertion and deletion procedure is the same as that of a chain; i.e., record insertion and deletion is done in the file area. However, the difference is that all changes to the status and linkages between records are done in the independent list area. The inverted list provides a very flexible response to user retrieval requests. This type of list provides for data retrieval on the basis of variable parameters. To illustrate this capability, assume that conditions A, B, C and D are the variable parameters to be used as the basis for record retrieval; i.e., only records that meet these conditions are to be retrieved. The file to be maintained for this example would include lists maintained for each condition. Upon request for the record(s) that meet(s) the given conditions, the different condition lists would be searched and its intersection would be the required record(s). Processing time to meet this kind of requirement would be much reduced if the condition that has the smallest number of records is searched initially, then the chosen records could be searched in the next smaller list and so on. This technique is often referred to as the least list principle. Another technique for reducing processing time is to numerically order the records for each condition and then apply the appropriate search algorithms.

The use of inverted lists has two principal advantages. The primary advantage of this approach is that complicated retrieval requests can be processed efficiently. Another advantage is that processing time for updating is much reduced

when compared to chaining, since most of the updating is done in the lists rather than in the file area. With this type of approach, no pointer is incorporated with the data in the record structure; thus, the reduction of record size could somewhat balance the storage space occupied by the lists.

Based on the aforementioned arguments for chaining and lists, the choice of an inverted list is warranted if selective data retrieval is frequent. This question is something the evaluator (with the guidance of the users) has to decide. A statistical analysis should be made of the frequency of retrieval of data elements prior to the selection of a data structure.

To go a step farther, one may take into consideration the possibility of using an involute type of data structure. An involute structure is a logical extension of the inverted list. It is a structure in which the data elements replace the pointers to records from the lists. The physical records are replaced by linkages of chains, each element of which belongs to a list. Each list would contain all the data elements belonging to it, hence elements could appear in any number of lists (up to its maximum). This structuring provides complete flexibility in file accessing and producing reports. However, the main arguments against this type of structuring are its increased processing time and its increased storage space requirement to maintain the lists.

B. STORAGE STRUCTURES AND FILE MEDIA TYPES

As previously mentioned, storage structure is the actual physical data organization in the system. Generally there are two possible ways of storing the data: sequential or non-sequential (random). The method of data storage is invariably dictated by the type of access that may suitably be adapted after selecting the data structure design. Three types of access may be sequential, indexed-sequential and direct. Sequential access does not lend itself to random storage of data, since the retrieval of records is based on some pre-designed logical sequence. For indexed-sequential access, sequential storage of data is applicable. The index and actual records are sequentially stored. Direct access of data for retrieval is accomplished by using an algorithm which converts between the record key and physical address.

The following is a tabular summary of storing and accessing of records:

TYPES OF ACCESS	TYPE OF STORAGE	
	SEQUENTIAL	RANDOM
Sequential	Yes	No
Indexed-sequential	Yes	No
Direct	Yes	Yes

The selection of file media is dependent on the storage structure. If random storage is desired, use of magnetic tapes is not feasible. Tapes could be considered as file media only if (1) the data are to be stored and accessed in

sequence, (2) the file is small and/or (3) the frequency of access is small.

The storage structure to be adapted should be a result of the users' requirements (speed, space, etc.) and cost analyses. Table I provides a comparative cost of a sample of storage units. It should be noted that the processor storage has the highest cost per byte and the magnetic tape and data cell have the lowest cost. Usually, the faster the storage unit, the higher the cost. For each type of storage unit the cost difference is usually directly proportional to the capacity.

TABLE I

COMPARATIVE COSTS OF STORAGE UNITS*

Quantity	Type	Model	Description	Capacity (in bytes)	Cost	Cost/byte
1 ea	2365	012	Processor storage	256 K	$401,000	$1.2500
1 ea	2301	001	Drum storage	4,000,000	80,700	.0202
1 ea	2820	001	Drum storage control		91,600	
1 ea	2311	001	Disc storage drive	7,250,000	21,390	.0030
1 ea	2841	001	Disc storage control		22,200	
1 ea	2314	001	Direct access storage facility	233,400,000	181,000	.0008
1 ea	2321	001	Data cell drive	400,000,000	111,000	.0003
1 ea	2403	001	Magnetic tape unit (2 drives)	56,000,000 (estimated maximum)	24,200	.0004
1 ea	2803	001	Tape control unit		26,500	

*Above IBM storage equipments are available at W. R. Church Computer Center, NPS. Costs are based on the IBM price list for fiscal year 1975.

III. ANALYSIS OF DATA STRUCTURING TECHNIQUES

Data organization is a term applied to the various methods of organizing data within a file. While Dodd's term is data organization, Lefkovitz calls it file organization. [House, 1974 and Lefkovitz, 1969]. There are various schemes of organizing data within a file. Dodd categorized all these schemes into three basic types of organization: sequential, random, and list.

From the preceding section four distinct types of data structures can be related to these data organizations' basic classifications offered by Dodd. They are: multiple record file, chaining, simple list and inverted list. Sequential data organization, as Dodd defined it, is associated with a multiple record file. Chaining, simple list and inverted list would be included in Dodd's list classification of data organization. Three data structuring methods are analyzed in this section: multiple record file, simple list and inverted list. They are used to describe the different data organization techniques. It is evident from Section II that data structure is a factor of prime importance in the evaluation of the data base management systems.

In evaluating data base systems, various criteria have to be considered. Different authors give different sets of criteria. The following set of criteria is deemed comprehensive and the most important one: retrieval capability, maintenance

capability, storage requirements and accuracy. The analysis of the three data structure methods is based on these criteria.

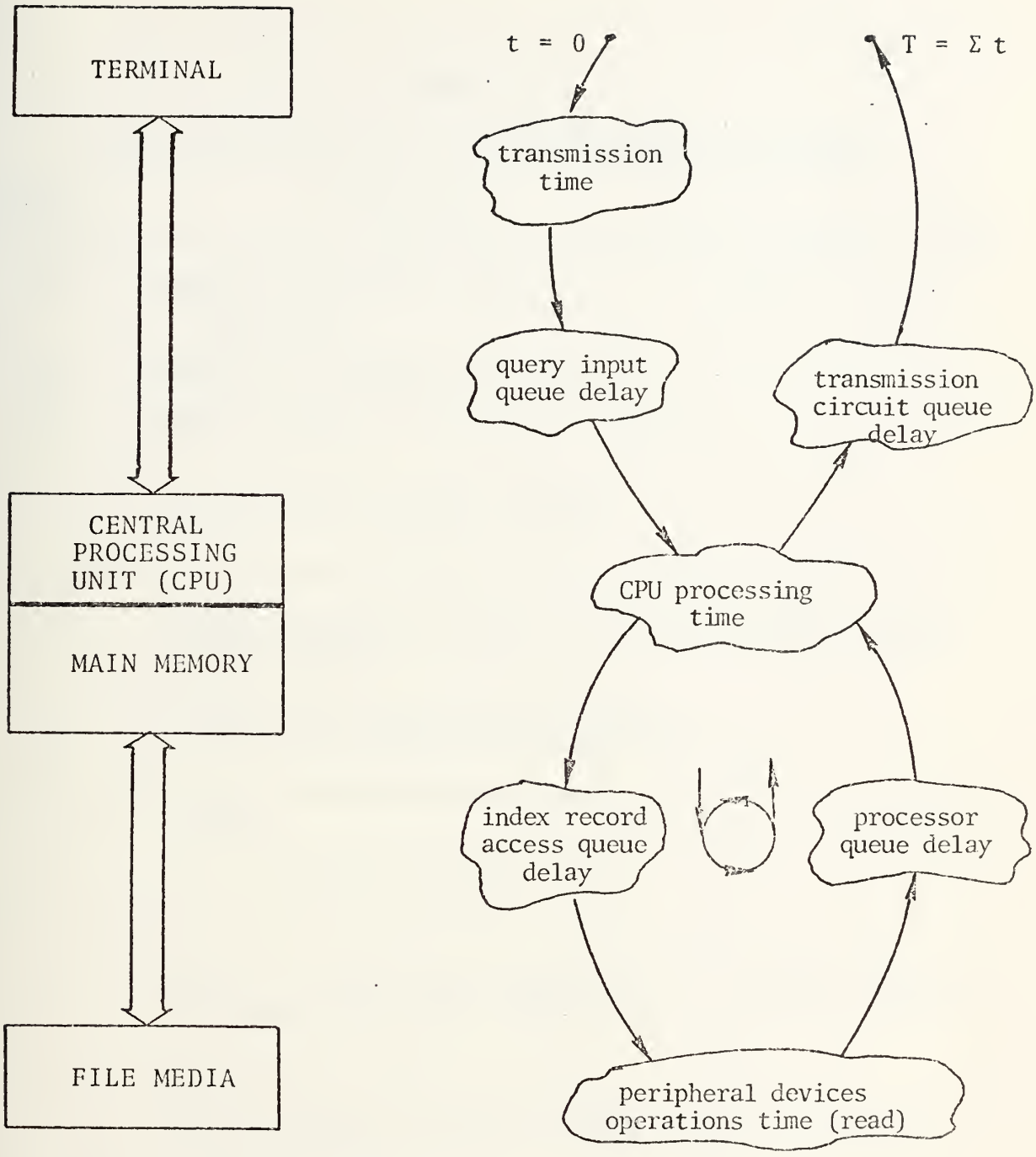
A. RETRIEVAL CAPABILITY

Figure 7 is a representation of the time involved with data retrieval. Time delays can occur between the central processing unit and the file storage, depending on the data structure type, available memory and index size. This will be explained later. Other factors contributing to the time duration between query and response from the system are not considered because they are not relevant to the analysis of the data structures. One of these contributory factors is "handshaking," a term used to designate the process of synchronization between the terminal and the computer when a query is logged in.

In the analysis we are involved only with the CPU processing time and storage units time. These are the time elements that vary with the data base configuration. The parameters in the time formulation are related to these factors.[2] Table II is a listing of these parameters, definitions and symbols. The file- and query-related parameters are dependent on the data structure. The device-related parameters are dependent on the file media characteristics.

The file-related parameters are: the total number of distinct keys, total number of records in the system, average number of keys per record, average length of lists and average

[2]The parameters and time formulation to follow are mainly by Lefkovitz. In some instances, a different terminology is used.



t ≡ time increment variable
T ≡ retrieval time
(portion of it)

PORTION OF DATA RETRIEVAL TIME TRACE

Figure 7

TABLE II

SYMBOL DEFINITIONS OF FILE PROCESSING PARAMETERS

Symbol	Definition	Parameter Type
V	Number of distinct keys in vocabulary	File related
N_r	Number of records in system	
N_k	Average number of keys per record	
L	Average list length = $\frac{N_r N_k}{V}$	
N_t	Average number of keys in a single query product	Query related
N_p	Average number of non-negated terms in a single query product	
L_s	Average shortest list length in query	
ρ	Ratio of query response to L_s	
A	Number of file record addresses per media physical record	Device related
T_r	Random access time of media	
R	Rotation time of media (direct access only)	

number of characters per record. The keys are the pointers listed in the index or indices for the list-type of data structure. The average of the number of keys per second is computed over all the pointers to each record.

The query-related parameters are: average number of keys in a single query product, average number of non-negated keys in a single query product, average shortest list length per query and the average ratio of query response to the average shortest list length per query. The term "query product" as used here refers to the record retrieved as the result of the query. The non-negated keys are the keys not eliminated from consideration in the process of searching record pointers within the index.

The file media-related parameters are: number of file record addresses per file media record, random access time of file media, transfer rate of file media, and for direct access storage devices (DASD), rotation time.

Excluding query interpretation time and the index directory decoding time, the list search retrieval times are: (1) for a simple list:

Time = List search and record transfer time

and (2) for an inverted list:

Time = List intersection time + List search and record transfer time.

Using the symbols, the formulae are: (1) for a simple list:

$$T_{SL} = L_s(T_r + 1.5R)$$ [3]

and (2) for an inverted list:

$$T_{IL} = \frac{L}{A}(N_t)(T_r + 1.5R) + (T_r + 1.5R)(L_s)(\rho).$$

1. CPU Processing Time

The CPU processing time varies with the data structure type. For a multiple record filing scheme, the CPU processing time of a query may not be significant. Upon receipt of the query, the CPU would determine which file is to be processed and the storage unit would be accessed and reaccessed until the desired record is found. For the list-type of data structure, the processing of the query would be more involved.

Many factors affect this interpretation process of the query by the CPU. One of prime importance is the index and file partitioning technique. When partitioning the indices of a complicated list data structure, the frequently accessed indices should be stored in fast DASD. In this way, the summation of indices access time is reduced. The same consideration applies to the partitioning of the records of the file among the file media available.

Another factor is the index size. The index must be accessed and transferred to the main memory if it is not in main memory. If the index is large and cannot be contained in the available portion of the main memory, a succession of

[3]The multiplier 1.5 of R is arrived on the assumption that a physical record occupies one track. Otherwise this multiplier would be the sum of the average track rotational delay (.5) and the fraction of the track occupied by a physical record.

accesses and searches of the index may have to be done before the right pointer to the record is determined. For a simple list and an inverted list of the same number of records, the inverted list is likely to have a larger index size because the number of pointers to each record is usually more than one. For a simple list, there is only one pointer for each record.

If the file involves more than one degree of inversion, the sequence of index access and search is also important. Usually, the shortest lists should be accessed and searched first in order to minimize the total length of the index to be operated on.

For a complicated list, the associativity of the records is another factor that affects the time duration of query processing by the CPU. If the records have minimum association, i.e., only one or a few keys are related to each record, then the fewer indices need to be intersected. This implies that less time would be required to retrieve a record from the data base.

Record selectivity is another factor. The file is considered to be highly selective if each key points to only one record in the data base. A completely inverted list, therefore, is of highest selectivity. In this case, while the time for list intersection would be longer, the search and access process of the record would be shorter than for the simple list.

2. Storage Unit Operating Time

The type of media that is used will affect the time duration between query and response. Usually the indices and

the records of the file would be on the peripheral device(s). The choices are many: drums, discs, magnetic tapes, paper tapes, etc.

The main factors to be considered in selecting a file media are the cost of the equipment and its speed. It is important to note that new technology is forcing reduction in storage cost.

Table III is a sample of DASD timing factors. For typical magnetic tapes, random access time is one minute and serial access is about four milliseconds. See Table I for a sample of storage unit capacity and cost. Typical file media used today are:

(1) Card. Slowest storage unit but inexpensive.

(2) Paper Tape. Least expensive storage unit that could be used for direct communications to the computer.

(3) Magnetic Tape. Serially processed; inexpensive mass storage media for large files; could be used for very fast off-line data transmission.

(4) Small Drums. Small backing storage for high activity data or programs; may hold indices.

(5) Small Disc Unit. Interchangeable file media; fast for serial processing; requires seek time of about 50 to 200 milliseconds before access.

(6) Large Disc Unit. Larger capacity; access time about 40 to 180 milliseconds.

(7) Large Drum. Largest fixed storage presently available, about a billion characters; fast access time of about 30 to 50 milliseconds.

(8) Magnetic Card File. Interchangeable cartridges of cards; access time of about 235 milliseconds.

(9) Large Magnetic-Strip File. Large random access file; interchangeable bins of strips; access time of about 90 to 400 milliseconds; can be used for fast serial processing.

TABLE III

SAMPLE OF IBM DIRECT ACCESS STORAGE DEVICES

Model	DASD Type	Rotation (ms)	T_r (ms)
2301	Drum	17.5	--
1301	Movable head disk	34	120
2311	Disk pack (movable head)	25	75
2321	Magnetic strip, cell store	50	400
(Data cell)	(movable head)		350*
			500**

* Previous strip restored

** Previous strip not restored

(10) Large Core Memory. Most expensive; fastest access time; access time of about 8 microseconds. [Martin, 1967].

Disc units may be either removable or nonremovable packs. Removable packs have longer access time, lower recording density and lower cost per byte than nonremovable packs. Further, the former offers unlimited off-line storage capacity while the latter's capacity is limited to the on-line storage.

Partitioning of files among the file media is not the only technique available to the evaluator or designer of data base systems for maximizing the speed of file operations. Various techniques of reducing seek and access times of DASDs have been developed. The main approaches to reduce DASDs' operation times are the following: (1) Records that are usually accessed in sequence are selectively stored on the device. Record processing time after an access is computed. After accessing a record, rotation time is allowed equivalent to the computed processing time. In this way the next record can be read immediately and no unnecessary rotation of DASD is incurred. (2) Another algorithmic approach to reduce the device operation time is to time the rotation and availability of read heads (applicable only to multi-head DASDs). See pages 293 to 296 of Ref. 14 for above algorithms and examples.

3. Data Structure Types and Data Retrieval Time

To review the preceding discussion, we note that the causes of the differences in data retrieval times attributable to data structure are: the time spent by the CPU on processing of query, index and record and the storage unit operating time. For convenience, we would designate this portion of data

retrieval time as the data retrieval sub-time (T_{sub}).[4] In this respect, there are three data structure types that are under consideration: multiple record file, simple list and inverted list. The following formulations are based on a single record retrieval.

a. Multiple Record File and Data Retrieval Sub-time

Under the multiple record file type of data structure, once the query for a record is received and interpreted by the CPU, the search for the record will start. No index or indices have to be accessed and searched and intersected for the record address. Rather, a succession of accesses and CPU processing of records might have to be done before the desired record is found. This CPU processing is actually the comparison process of records. With this type of data structure, the records should be stored sequentially, especially if magnetic tapes are used. If records are randomly stored, random accessing of records has to be done.

When DASDs are used, other search algorithms are possible. One of these is the logarithmic or binary search.

b. Simple List and Data Retrieval Sub-time

Data retrieval sub-time for this type of data structure has been partially formulated in the previous discussion. Index decoding time, however, has to be added to the previous formulation. Sub-time would be

$$T_{sub} = T_{SL} + N_p T_{cpu} \quad \text{or}$$

---

[4] See Figure 7 for a complete picture of the data retrieval time.

$$T_{sub} = L_s (T_r + 1.5R) + N_p T_{cpu}.$$

Assignment of values for T_r and R are dependent on the storage unit average operation time (see Table III).

c. Inverted List and Data Retrieval Sub-time

Sub-time formulation for this data structure type has also been partially done. Like the simple list, the index decoding time has to be added. The resultant equation is

$$T_{sub} = T_{IL} + N_t T_{cpu} \quad \text{or}$$

$$T_{sub} = \frac{L}{A} (N_t) (T_r + 1.5R) + \rho L_s (T_r + 1.5R) + N_t T_{cpu}$$

or

$$T_{sub} = \{\frac{L}{A} (N_t) + \rho L_s \}\{ T_r + 1.5R \} + N_t T_{cpu}.$$

Also like the simple list, assignment of values for T_r and R are dependent on the storage unit operating times; sample values are listed in Table III.

Note that the T_{cpu} for the list structures would be different from T_{cpu} of the multiple record file. The T_{cpu} times are dependent on the number of machine instructions necessary to compare records and to decode the index.

B. MAINTENANCE CAPABILITY

In some data base systems, more time is expended on maintaining files than on searching them. Because of this situation, file maintenance speed and efficiency determine the cost and/or feasibility of the entire system. It is reported that 20 to 30 percent of total machine time is spent for the file maintenance function of sorting in general business applications. [Meadows, 1967].

Sorting and merging are the best known as well as the most common file maintenance functions. Another maintenance function is the initial file creation. The file maintenance functions can be subdivided into: (1) addition of records to a file, (2) deletion of records from a file, (3) changing the value of a record's field, (4) changing the record's structure, (5) changing record sequence in a file, and (6) changing the file storage media. The fifth function is sorting.

This subsection will describe the capabilities of the three general data structures--multiple record file, simple list and inverted list, in terms of the above functions. Simple and inverted list will be included under lists structures.

1. Maintenance of a Multiple Record File

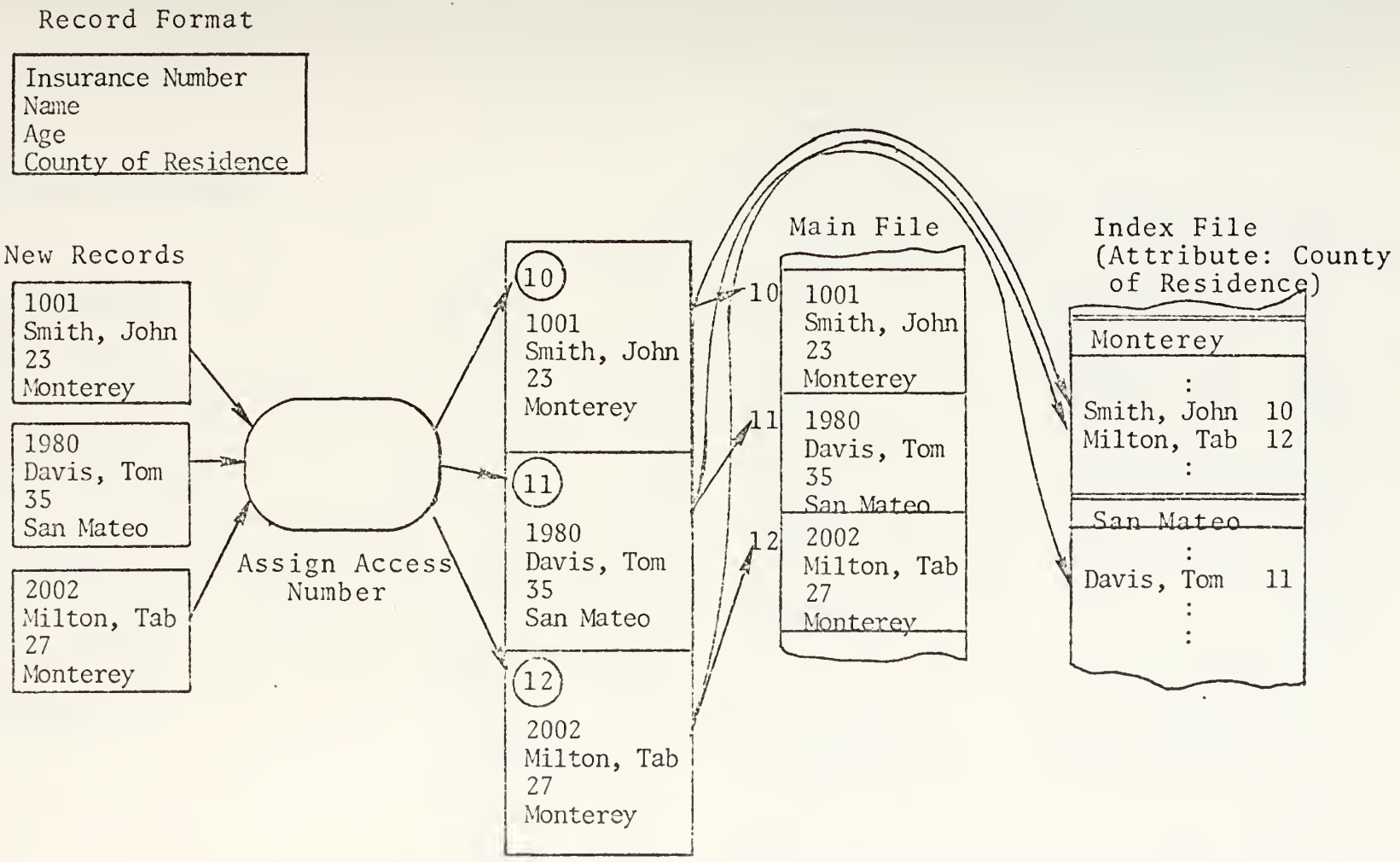
The maintenance of a multiple record file is trivial unless a sequence of records other than their arrival sequence is desired. If records are not sequenced, the new records are usually added at the end of the file. In this case no complex search for spaces for the records to be inserted is required. The situation is entirely different if a record sequence is maintained. Here oftentimes the entire file must be copied.

Deletion of records can be done in more than one manner with this data structure type. One method of deletion for this data structure is to flag the records to be deleted as such by changing a value of one of their fields. By this method the records to be deleted would not be physically removed from the file. Another method of deleting records is to copy the entire file, excluding the records to be deleted. The latter method

is necessary if space is needed for new records. The changing of records content or structure is done by changing, adding or deleting fields in memory. The fifth function (sorting) involves alteration of record sequence. Multiple passes through the field may be necessary to accomplish this. The last function, which involves changing the files medium, is considerably a simple process.

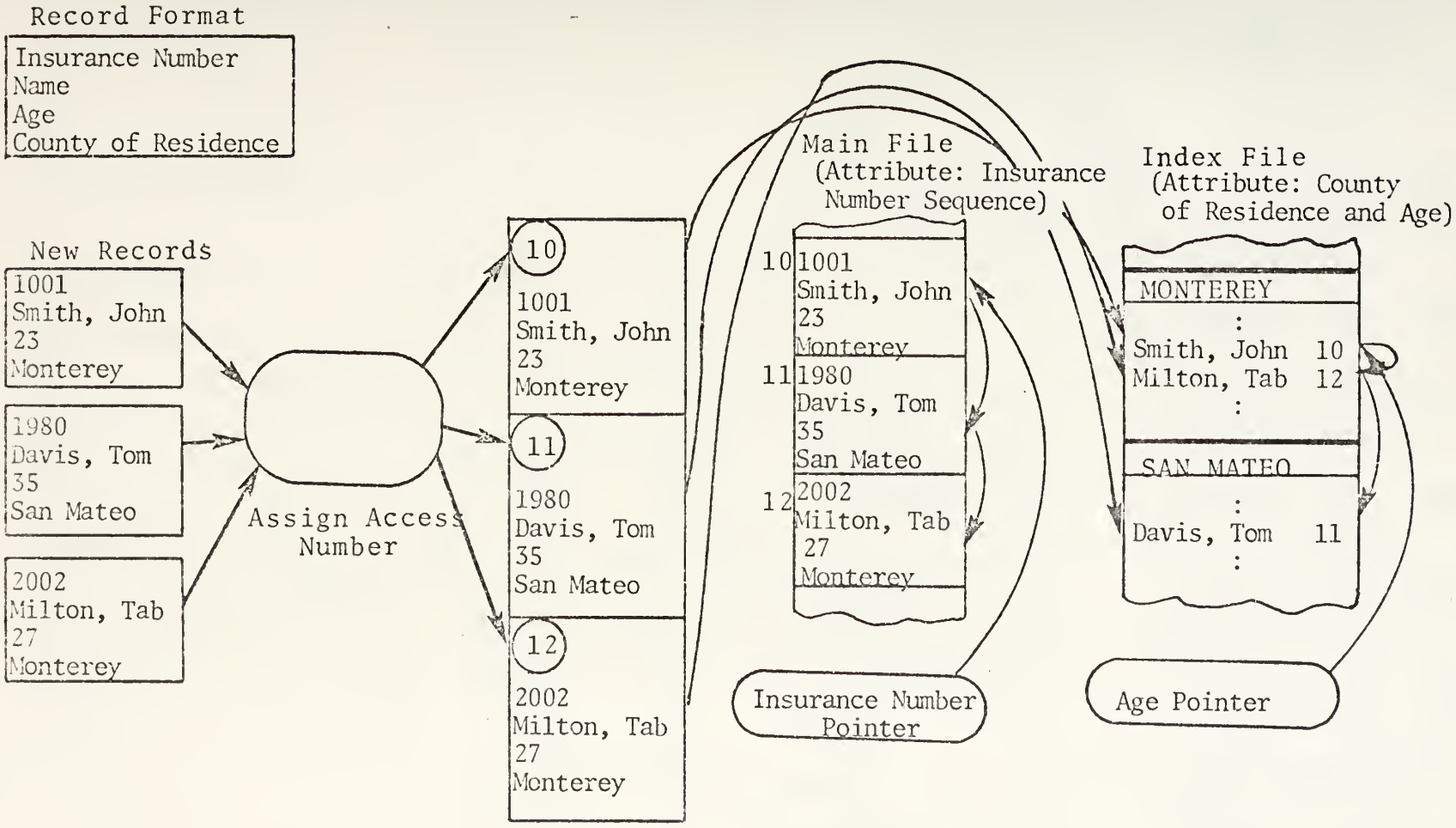
2. Maintenance with List Structures

Different procedures for the addition of records to a list data structure are diagramed in Figures 8, 9, and 10. Variations among the different procedures are dependent on the structure of the file to which the new records are to be added. The records may be arranged in sequence logically or physically within the main file. For example, in Figure 9 the new records are logically sequenced within the main file by means of pointers. In this figure, the insurance number and the age pointers are used to point to the numerical sequence of insurance numbers and age, respectively. The updating procedure for the main file using this scheme is complicated because the pointers must be updated. In Figure 10, the new records are physically sequenced by hundreds of insurance numbers per track of the disc. After placing the new records in the main file, the records are posted to the appropriate indices. This posting operation is a complex one. It requires either that the index files (indices) be copied as records are being added at many points throughout the file [Figure 8], or that chaining [Figure 9] or list structures [Figure 10] be used to permit



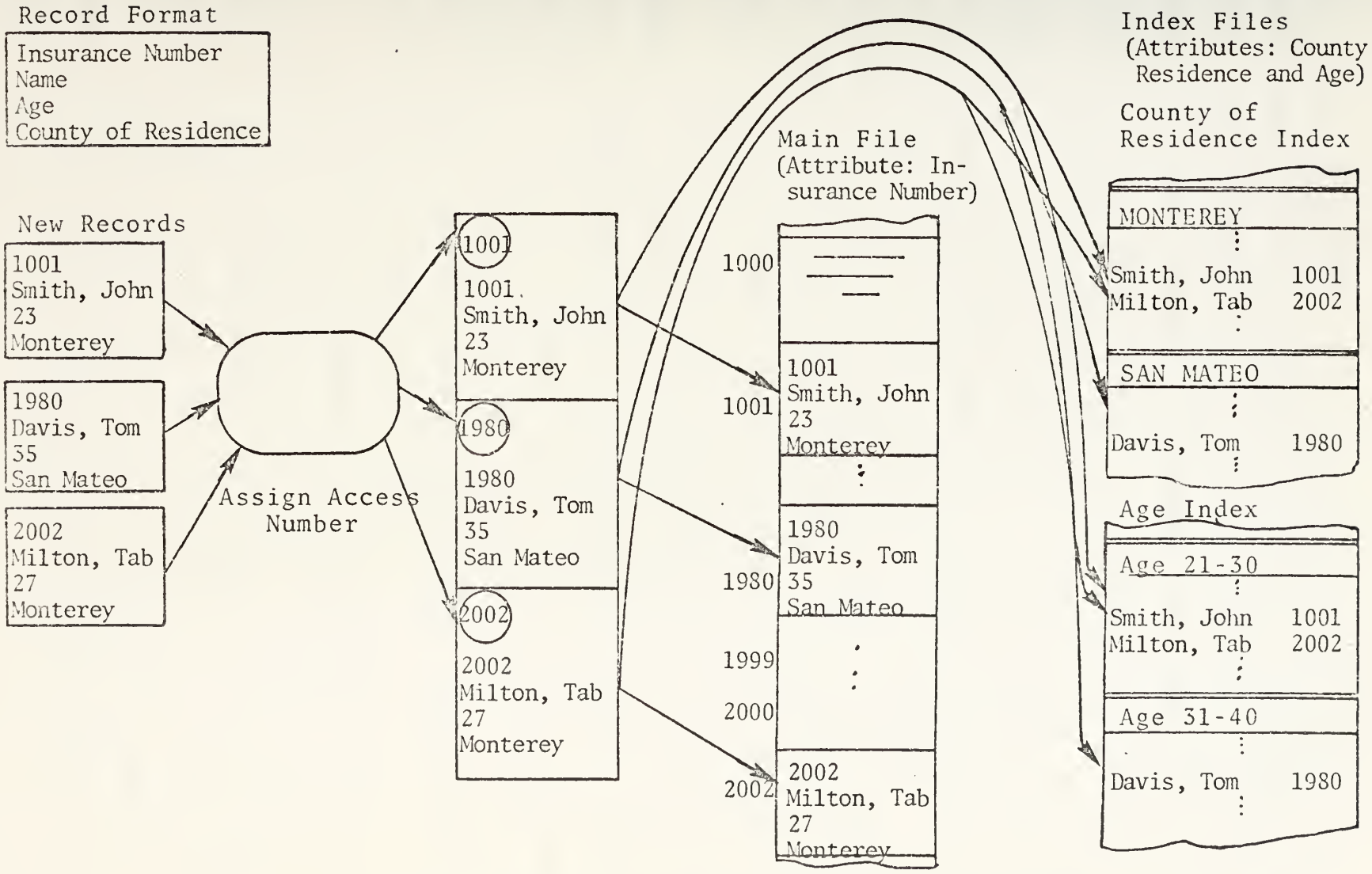
ADDITION FUNCTION OF MULTIPLE RECORD TYPE OF INDEX FILE

Figure 8



ADDITION FUNCTION OF CHAINED-TYPE OF INDEX FILE

Figure 9



ADDITION FUNCTION OF LIST-TYPE OF INDEX FILE

Figure 10

noncontiguous placement of records in the index files. The use of chaining within the index files is a complicated scheme because the succeeding record pointers have to be accessed and altered for every record that is added. The complexity of the process increases as the degree of file inversion increases.

The main advantage attributed to list file, with regard to addition of records, is that the logical sequence of records can be continuously maintained. The copying of files, which is often necessary for multiple record files, is avoided. At most, only the index files have to be copied. For a simple list there is only one index. However, for inverted lists the number of indices that may need to be copied can be large.

The process of record deletion with list files is also complex. If chaining is used within the main file or the index files, a complicated process of updating the pointers would be required. A further problem involves the accounting for spaces available to the records within the entire file. This would increase the processing time and grab spaces from the addition or deletion process during execution.

The process of changing the record values would be a simple and fast process with list files. Pointers could be traced to the required records. The only factor that could cause prolonged search time for the records are pointers that span a large portion of the file. In such cases, numerous transfers of records between main and auxiliary memory would occur.

A record sequence change is the most difficult maintenance function of list files. Many accesses to records and alterations of pointers may be necessary for relatively few changes in sequence. The last function of changing the file medium is a simple process; it involves copying the file from one medium to another.

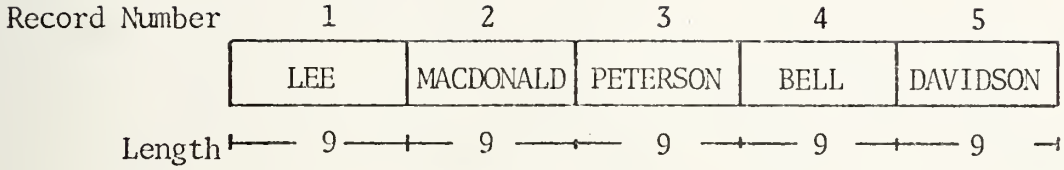
C. STORAGE REQUIREMENTS

Another criterion is storage requirements. Auxiliary storage requirements are analyzed in this subsection.

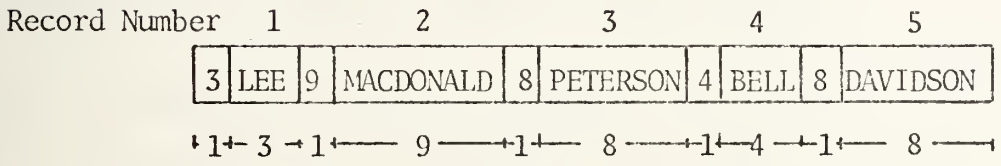
1. Auxiliary Storage Requirements of Data Structures

There are three basic methods of storing records on auxiliary devices. They are: sequential records, indexed records, and chained records. In sequential storage, records relationships are based on record adjacencies. In indexed record storage, indices are used to relate records to one another. In chained record storage, related records are chained together.

Figure 11 depicts sequential record storage. Figure 11(a) is a sequential storage of fixed length records. Note that 14 bytes of spaces are wasted. Figure 11(b) is a sequential storage of variable length records. The total spaces required are 37 bytes, a saving of 8 bytes. The difference would be great as the number of records increases. The situation could be the opposite though. This would occur if the records could be made of the same length. For example, if dates would be represented as numbers in records, say "010275" for January 2, 1975, "050475" for May 4, 1975, then it is better to use the fixed length records.



(a) Sequential Storage of Fixed-Length Records (all records are 9 bytes long)



(b) Sequential Storage of Variable Length Records

SEQUENTIAL STORAGE OF RECORDS

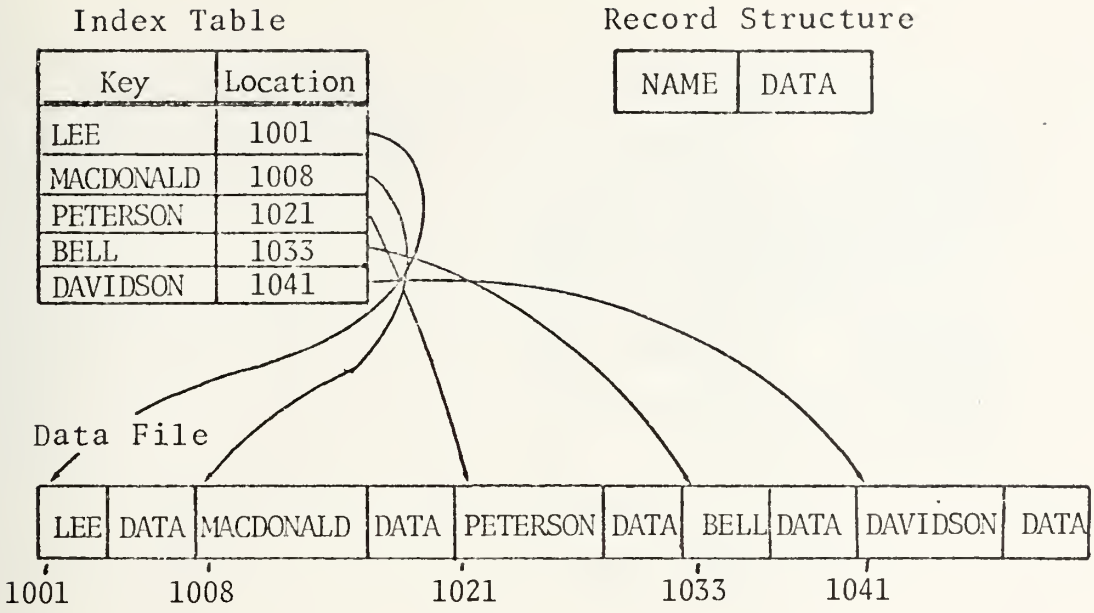
Figure 11

In Figure 12 are diagrams of indexed records storing schemes. Figure 12(a) is a single index record storage scheme. Figure 12(b) is a multiple index record storage scheme. Note that in Figures 12(a) and 12(b) the use of fixed length records would add to the space requirements. Compared to the sequential storage of variable length records, additional space is required for the indices. The additional space required would increase with the degree of file inversion. Possibly the index tables may exceed the size of data. This is the price one must pay for the capability to relate records using list files.

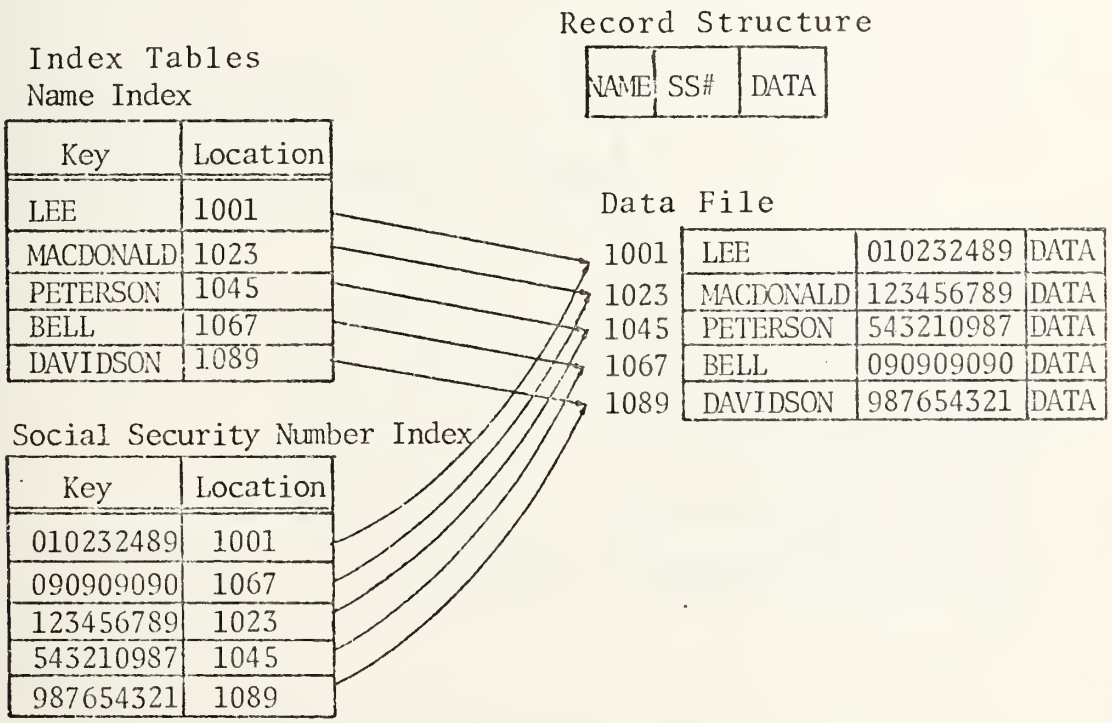
Chained record storage requires more space than sequential record storage. Added space is required for the record pointers. Compared to index records storage, this scheme requires no space for an index. Figure 13 depicts chained record storage for a single and multiple chains. In this figure the pointers are used to follow the numerical sequence of the attributes of social security number and age.

D. ACCURACY OF DATA RETRIEVAL

Accuracy can be defined as the degree of data retrieval correctness subject to some degree of effort exerted. This data retrieval accuracy concept can be subdivided into what is commonly known as the recall and precision ratio. The recall ratio is defined as the degree of success in retrieving relevant data from a relevant system. Recall ratio is the system's capability to let through the desired data. The precision ratio is the system's capability to hold back unwanted data. Quantitatively, these ratios could be expressed as:



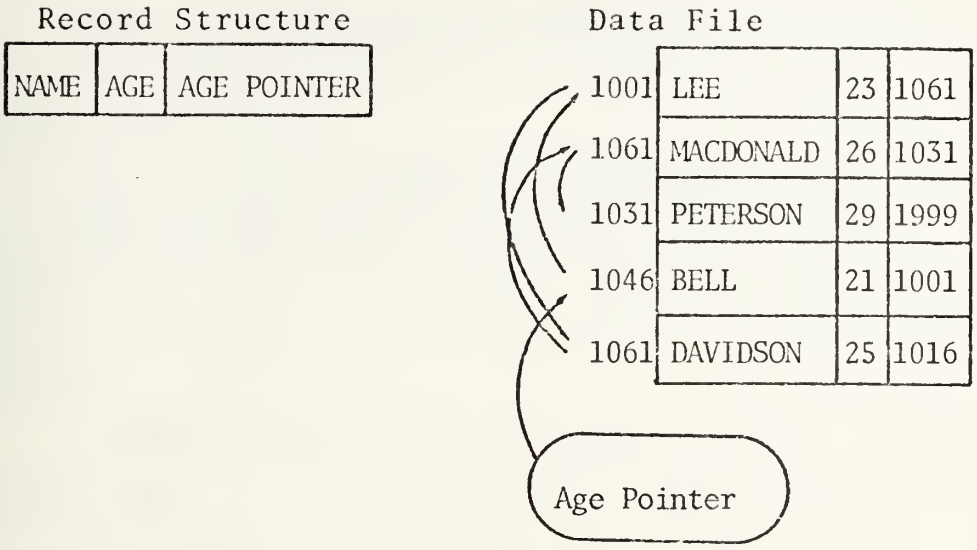
(a) Single Index Record Storing (simple list)



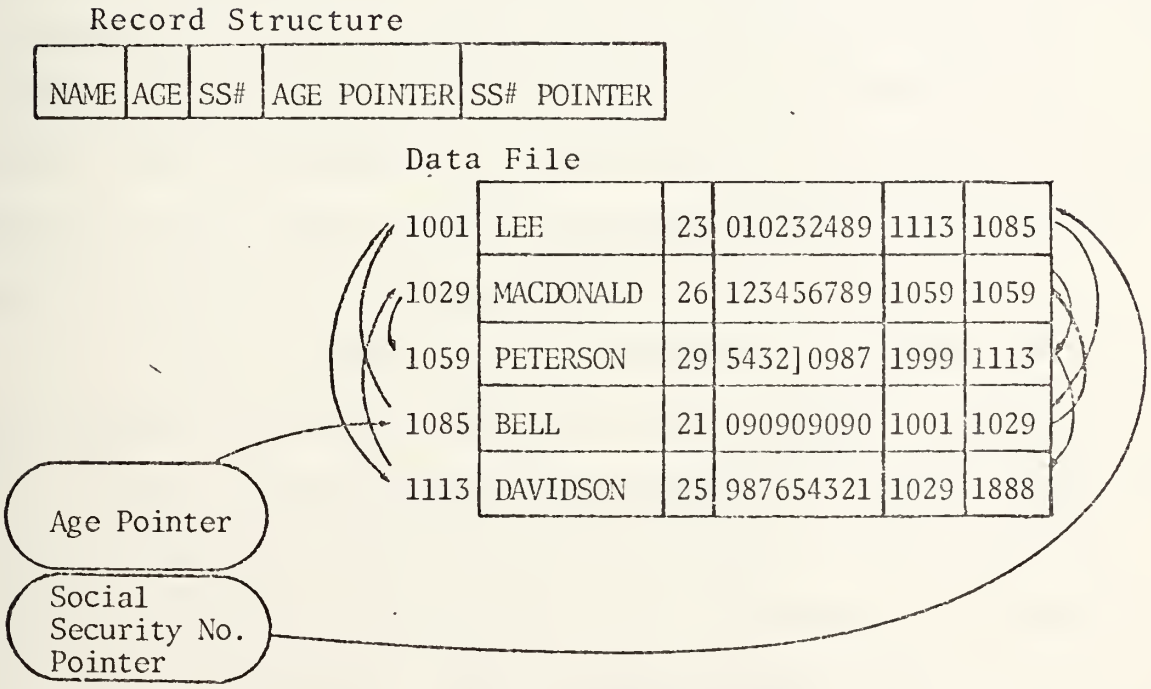
(b) Multiple Index Record Storing (inverted list)

INDEXED RECORD STORING

Figure 12



(a) Single-Chain Records Storing



(b) Multi-Chain Records Storing

CHAINED RECORD STORAGE

Figure 13

Recall Ratio =

$$\frac{\text{Number of Relevant Data Items Retrieved by the System}}{\text{Total Number of Relevant Data Items in the system}} \times 100$$

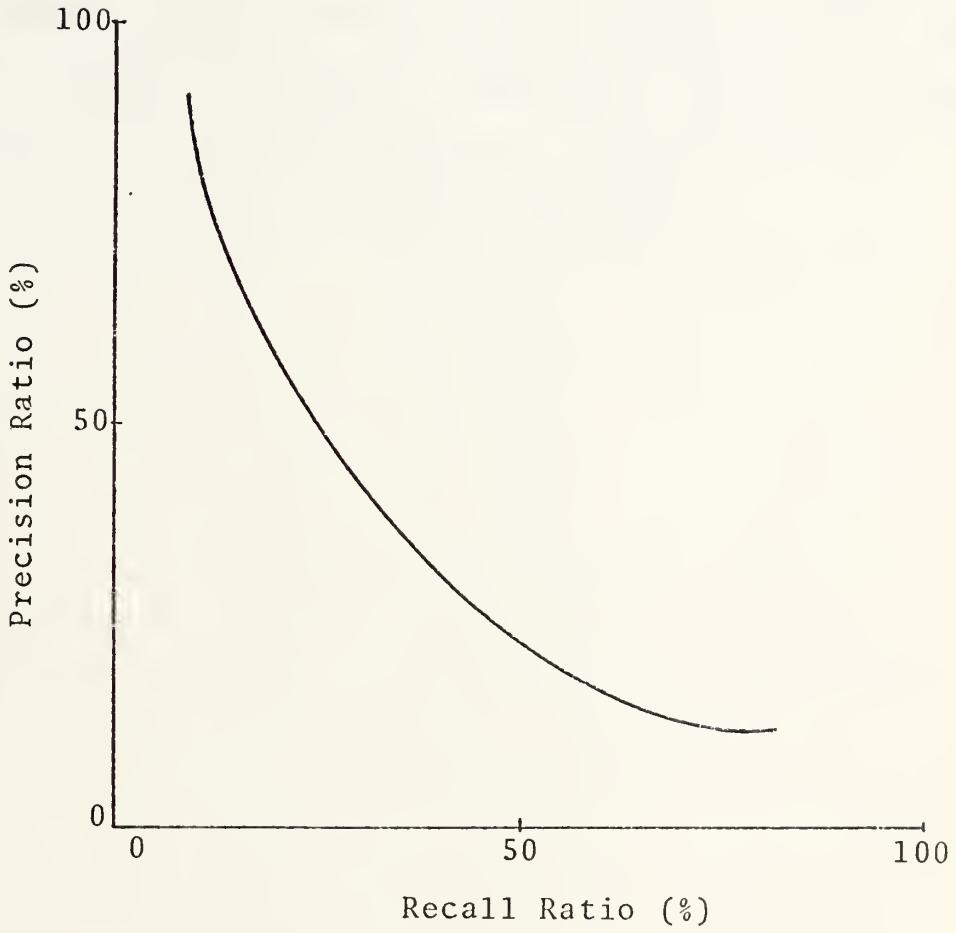
Precision Ratio =

$$\frac{\text{Number of Relevant Data Items Retrieved by the System}}{\text{Total Number of Data Items Retrieved by the System}} \times 100.$$

To fully understand these ratios, the following example is considered:

Suppose there are 30 relevant records in an inquiry to a data base. A system search is conducted and 25 of these relevant records are retrieved. The recall ratio is 25/30, or about 83%. Assume that another 20 irrelevant records have been retrieved with the 25 relevant ones in the process. The total retrieved records are 45, hence the precision ratio is 25/45, or about 56%. It is usually stated that the search has a 83% recall at a precision of 56%. It has been observed that as the recall ratio increases the precision ratio decreases: A typical plot of recall ratio versus precision ratio is shown on the next page. [Lancaster and Fayen, 1973].

Data structure type is certainly a relevant factor in determining a system's data retrieval accuracy. Unfortunately, this is not the only factor that affects the system's accuracy. Size of data base and volatility of data are other factors. Due to the multiplicity of factors, it is difficult to determine the accuracy of retrieval. One may approximate the system's accuracy by simulation. Simulation should, however, be the last recourse.



RECALL RATIO (%) VS. PRECISION RATIO (%) PLOT

Figure 14

E. TRANSIENCY

Transiency of a data base is its tendency to be disorganized. The more transient a data base, the more time and processing would be required for its reorganization. Data structure type affects the system's transiency. Logically, transiency would increase with the degree of inversion. Transiency would also increase with the degree of chaining utilized.

IV. SUMMARY

In data base system selection, one may hope for but should not fully expect a perfect system. The circumstances impacting on the evaluator or designer would considerably affect the selection decision, e.g., financial and political considerations. Another factor is the possible contradiction in users' requirements. A case in point is a requirement for a data base system with a powerful search capability with minimum storage space. Finally, technological progress is a factor. As one completes a large data base system design, developments may occur which could be utilized in the system design.

Various topics regarding data base systems have been discussed in this paper. They involve current practices in this field. Possible tradeoff considerations have been emphasized where they are important. The central topic in this paper is data structures. This subject area has been emphasized because it is important with respect to the selection of storage structures, file media and file maintenance functions. This implies that one can judiciously select data base systems by placing great emphasis on data structure considerations.

Four categories of data base criteria have been analyzed: retrieval capability, maintenance capability, storage requirements and accuracy of data retrieval. It is by no means claimed that these criteria are complete. Using these criteria, however, could provide some assurance of the correctness of the data base evaluation.

BIBLIOGRAPHY

1. Alexander, M.J., Information Systems Analysis, Science Research Associates, Inc., 1974.

2. Coffman, E.G.,Jr. and Denning, P.J., Operating Systems Theory, Prentice-Hall, Inc., 1973.

3. Conference on Data Systems Languages Systems Committee, A Survey of Generalized Data Base Management Systems, May 1969.

4. Conference on Data Systems Languages Systems Committee, Feature Analysis of Generalized Data Base Management Systems, May 1971.

5. Elson, M., Data Structures, Science Research Associates, Inc., 1975.

6. Flores, I., Data Structure and Management, Prentice-Hall, Inc., 1970.

7. Hartman, W., Matthes, H., and Proeme, A., Management Information Handbook, McGraw-Hill, 1968.

8. House, W.C. (editor), Data Base Management, Pretrocelli Books, 1974.

9. Knuth, D.E., The Art of Computer Programming, v. 3, Addison-Wesley Publishing Co., 1973.

10. Krauss, L.L., Computer-Based Management Information Systems, American Management Association, Inc., 1970.

11. Lancaster, F.W. and Fayen, E.C., Information Retrieval On-Line, Melville Publishing Co., 1973.

12. Lefkovitz, D., File Structures for On-Line Systems, Spartan Books, 1969.

13. London, K.R., Techniques for Direct Access, 1st ed., Auerbach Publishers, Inc., 1973.

14. Madnick, S.E. and Donovan, J.J., Operating Systems, McGraw-Hill Book Co., 1974.

15. Martin, E.W.,Jr. and Perkins, W.C., Computers and Information Systems: An Introduction, Richard D. Irwin, Inc. and The Dorsey Press, 1973.

16. Martin, J.T., Design of Real-Time Computer Systems, Prentice-Hall, Inc., 1967.

17. Meadow, C.T., The Analysis of Information Systems: An Introduction to Information Retrieval, Wiley, May 1967.

18. Snuggs, M.E., Popek, G.J. and Peterson, R.J., "Data Base Objectives as Design Constraints," Data Base, v. 6, No. 3, pp. 11-20, Winter 1974.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 72 Computer Science Group Naval Postgraduate School Monterey, California 93940	1
4.	Professor N. F. Schneidewind, Code 55 Ss (Thesis Advisor) Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
5.	LCDR Rodolfo M. Clautero Philippine Navy (Student) 56 E. De La Paz Street Angono, Rizal Philippines	1
6.	Naval Computer Center Sangley Naval Station Sangley Pt., Cavite City Philippines	1